Symlink

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-24

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 5310 bytes

Attack Category	• Path spoofing or confusion problem
Vulnerability Category	• Indeterminate File/Path • TOCTOU - Time of Check, Time of Use
Software Context	• File Management
Location	
Description	The symlink() function creates a symbolic link. It is generally vulnerable to classic TOCTOU attacks. A call to symlink() should be flagged if the first or second argument is used earlier in a check-category call.
APIs	**Function Name** / **Comments** symlink /
Method of Attack	The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results. The symlink() call is a use-category call, which when preceded by a check-category call can be indicative of a TOCTOU vulnerability. A TOCTOU attack in regards to symlink() can occur when a. A check for the existence of the file occurs or a non-fd reference (pathname) to the filename occurs b. An actual call to symlink occurs. Between a and b, an attacker could, for example, link the referenced file to a known file. The subsequent

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

	symlink() call would have an unintended effect or impact.
Exception Criteria	
Solutions	(see nested table below)

Solution Applicability	**Solution Description**	**Solution Efficacy**
Generally applicable to any symlink.	Utilize a file descriptor version of stat/ fstat when checking.	Effective.
Generally applicable to any symlink.	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.
Generally applicable to any symlink.	Limit the interleaving of operations on files from multiple processes.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
Generally applicable to any symlink.	Limit the spread of time (cycles) between the check and use of a resource.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.

	Generally applicable to any symlink.	Recheck the resource after the use call to verify that the action was taken appropriately.	Checking the status after the operation does not change the fact that the operation may have been exploited but it does allow halting of the application in an error state to help limit further damage.

Signature Details	int symlink (const char *oldname, const char *newname);

Examples of Incorrect Code	`int check_status;`
	`int use_status;`
	`struct stat statbuf;`
	`...`
	`check_status=lstat(oldname, &statbuf);`
	`...`
	`use_status=symlink(oldname, newname);`
	`...`

Examples of Corrected Code	`...`
	`if (symlink(oldname, newname) != 0)`
	`printf("Error creating link\n");`
	`...`

Source Reference	• ITS4 Source Code Vulnerability Scanning Tool [2]
Recommended Resources	
Discriminant Set	**Operating System**: • UNIX (All) **Languages**: • C • C++

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com